(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
8 August 2002 (08.08.2002)

PCT

(10) International Publication Number
WO 02/061613 A2

(51) International Patent Classification⁷: G06F 17/30

(21) International Application Number: PCT/EP02/01027

(22) International Filing Date: 1 February 2002 (01.02.2002)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
101 04 831.9 1 February 2001 (01.02.2001) DE

(71) Applicant *(for all designated States except US)*: **SAP AKTIENGESELLSCHAFT** [DE/DE]; Neurottstr. 16, 69190 Walldorf (DE).

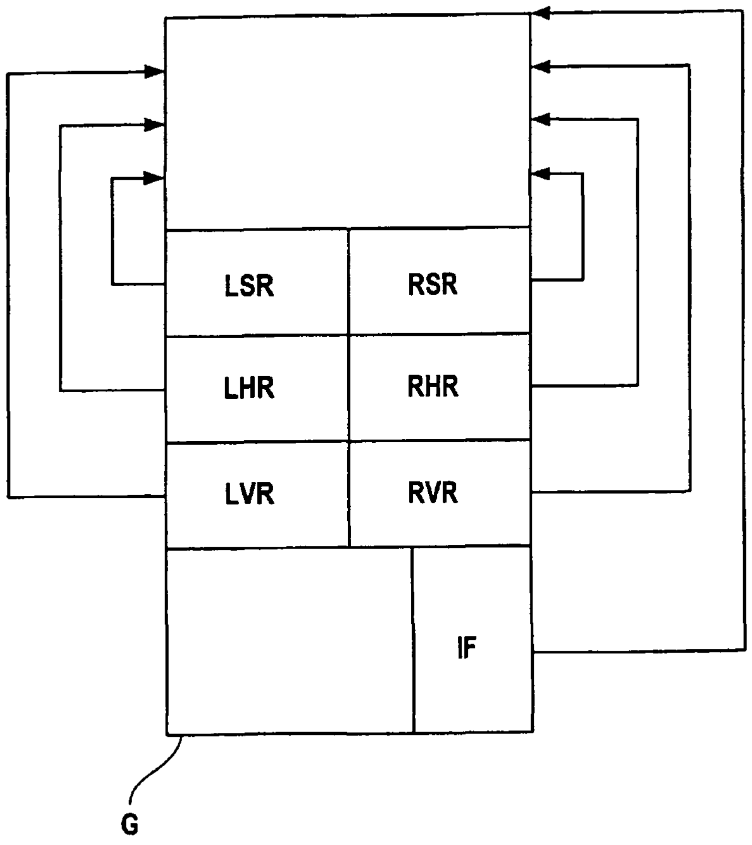
(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: **VON BERGEN, Axel** [DE/DE]; Breslauerstr. 27, 69168 Wiesloch (DE). **SCHWARZ, Arne** [DE/DE]; Hildastr. 2, 69115 Heidelberg (DE). **SAUERMANN, Volker** [DE/DE]; Keplerstr. 24, 69120 Heidelberg (DE).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent

[Continued on next page]

(54) Title: DATABASE SYSTEM AND QUERY OPTIMISER



(57) Abstract: Method for optimising a query with respect to a database structure, comprising receiving database information with respect to a specific data source, generating a search database structure for said data source, based on said received data source information, receiving a query request for said data source, analyzing the search database structure by counting a number of hits for respective sections of said query request, and calculating an optimized query request based on the analysis performed.

WO 02/061613 A2

(BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

Database system and query optimiser

This application claims priority from the German patent application DE 101 04 831.9, the text of which is herewith incorporated by reference.

The invention relates to information storage systems, such as database systems. In such computer based data storage systems, data is stored in and retrieved from some medium, such as a memory and a hard disk drive. The most common way to approach a database is via a database query, for example in the form of a SQL statement. Such a database query is practically always in a compound form, i.e. that it requires at least two conditions to be fulfilled. A search for data that conforms to the query can be done in a number of ways. Often a query optimiser is used, for example a rule based optimiser or a cost based optimiser. The rule based optimiser uses a set of predefined rules. The cost based optimiser uses statistical information about the data to structure queries, by estimating the selectivity of each query component and leading the search path along the most selective components first. Even though using the known optimisers can shorten the average response time, the need exists for optimisers that optimise each specific query.

In known systems, the data is stored in a data structure such as for example a structure based on the relational model. Although data stored using the known data storage systems can be stored, searched and retrieved, the time required for the retrieval can be considerable, especially in case of complex queries. Furthermore, constructing an efficient database structure is complicated and therefore costly. If the design of the database is not up to par, more or less severe performance penalties can easily result. Also, once a database layout or data model is defined, later alterations are difficult to implement, and almost always lead to loss of performance. Therefore the need exists for a database system that can be implemented and altered without introducing unwanted performance restraints.

The first goal is met by providing a query optimiser according to claim 1. By basing a search path decision on a number of hits for each section of a compound query, a search can be guided through the least number of hits first, thus increasing the chance for a quick return of a result for the query. By providing the search structure with additional links that represent data record relations according to claim 4, the optimiser can not only take in account the number of hits of a certain query section but also distil those keys for the search that are most selective.

By providing a data structure according to claim 6, a database structure is achieved that can be implemented without performance restraints, and can be used with the abovementioned query optimiser method. Later alterations are easily achieved without risk of sacrificing performance. By using the data structure of the invention, all advantages of the query optimiser are integrated within the data structure itself. This eliminates the need for a separate search data structure.

Additional features and advantages will become apparent in the following description of a number of embodiments of the invention. Advantageous variants of the invention are subject of the dependent claims.

Fig. 1 shows a database with a table,
Fig. 2 shows an example of a tree structure according to the invention,
Fig. 3 shows another example of a tree structure according to the invention,
Fig. 4 shows an example of a data structure according to the invention,
Fig. 5 shows a detail of a data structure according to the invention, and
Fig. 6 shows schematically a data element according to the invention.

In fig. 1 a first example of an embodiment of the invention is shown. A data source DS (which can be for example a database or an application) containing data is shown, with a small portion of stored data in the form of a table C with columns A and B, comprised of data elements. Note that the data source DS can be of any kind, and that the table C is shown as an example.

According to the invention, an external data structure E is formed as shown in fig. 2. For convenience the data structure E is called a search data structure, as it will be used to perform searches on it to analyse the structure based on a data query. In the data structure E, the elements of the respective columns A and B are organised in a binary tree structure. The elements of the trees are shown as circles in the fig. 2, with

in the upper left part the value of the attribute (VAL), in the upper right part a unique identification of the type of element (ID) and in the lower part a number (COUNT) that at least represents how many elements are present in the branches below the respective element. Note that in this example the lower number is inclusive of the respective element itself. If in the column A or B, multiple entries exist for a certain instance, then these are accounted for by increasing the number COUNT with the number of respective entries. When elements are added or deleted from the tree, the COUNT numbers have to be updated in the tree.
The data can be transferred from the data source DS to the implementation of the program of the invention in any suitable way, for example via a data communication link, such as the Internet, and the data structure to the invention can be build in any suitable way. Preferably, the search data structure is stored in a memory that can be randomly accessed, such as for example RAM, which gives fast access times.

In this example, only a very small section of a data source DS is shown. In practice, a great number of columns are used (and therefore also a corresponding number of trees), and similar a great number of tables or similar constructions are used. Furthermore, the trees will be in the implementation accessible through some means, for example a linking element such as a pointer, that is positioned in hierarchy over the tree structures. The linking means itself can be part of a further configuration to facilitate access to the tree structures.

When a query for a search in the data source DS is requested in the form of a compound query for example Q1 AND Q2 AND Q3 (wherein the conditions Q1, Q2, Q3 are sections that are each a more or less selective statement for a search in the database), a query optimiser according to the invention will conduct a search on the data structure E as follows. For each section of the query (i.e. Q1, Q2, Q3), the number of hits is determined by descending along a path the respective braches of the tree A, B until the required element or elements have been found, and the number of hits can be calculated from the respective values COUNT. The number of elements that meet the criterion can be obtained straightforward from the COUNT parameter of the element found itself, or by simple addition and/or subtraction of multiple elements found. It is therefore not necessary to traverse the complete tree to the end to obtain the data required. Note that searching through a (binary) tree as such is known in the art, as well are searches for ranges in a (binary) tree.

As a result the number of hits for each respective component Q1, Q2, Q3 are known; based on this information the optimiser can select an order in which to execute the query, preferably starting with the component that has the lowest number of hits, as this is potentially the most selective condition. The number of hits per component can also be used in any other way for optimising, including combining or using it together with other criteria.

In this example it is not required that the structure E is updated in real-time. Depending on the rate with which the data changes in the data source DS, the structure E can be updated periodically, like for example hourly or daily. By not having a fully up to date structure E, an error is introduced as not the exact actual number of hits for the data source DS is calculated, but if the error is kept within predetermined ranges the estimate can be effectively used. A periodical update has the advantage that it does not require so many resources as a real-time update would cost.

In a further embodiment of the invention, not only the number of elements, but also the relations between elements is included into the structure, now shown in fig. 3 as E'. The linking elements H are shown by dotted lines and represent the data as incorporated for example in a record; the dotted line represents the connection between the fields of a record. The linking element can for example be implemented as a pointer. A further tree F is shown, this tree represents a further column of the table C. As is shown in fig. 3, the subsequent columns of the table C are arranged in a tree structure, being sorted over the respective ID numbers. This has the advantage that access to the respective trees A, B, F can be made very fast and efficient.

When analysing the query components, not only the number of hits can be calculated, but also the most selective keys can be found, wherein any key can be selected. The query optimiser would as result return query components in an optimum order, wherein the query components are not necessarily the same as those in the original query. With the new query, the data source DS can be searched. By using this implementation a quick and efficient way to get the optimum query keys is provided.

Preferably, every tree is identified by an integer value for its identification (ID), and also its elements are preferably identified by an integer or other simple identifier type. This has the advantage that during the search only relatively simple (and therefore fast) comparisons have to be made.
Note that in fact it is not essential to have a binary tree; the invention can also be applied for example with AVL trees, 2-3 trees, B-trees, and splay trees, and in principle any data structure that allows range searching. However, a binary tree (and in particular a balanced tree) promises the fastest overall access time, and is relatively easy to implement and use.

In the previous shown examples, the invention was used together with a separate data source. This has certain benefits, such as the fact that an already existing database system can be used, and the application can be used separately from the main data source. In a further embodiment of the invention, the search database of the second example is augmented with certain features to obtain a database structure in which data can be stored and retrieved, while incorporating the advantages of the fast query optimiser.

In fig. 4 a table K is shown with three columns, respectively first name, age, and weight. In the database structure according to the invention, each column is organised in a binary tree, preferably a balanced AVL tree. The tree is composed of cells that contain the data of the table. The connection between individual cells in the respective trees, that is the connection that makes up a line (or data record) in the table, is made via the structure H'. This structure H', for example made out of pointers, connects the respective cells of neighbouring trees. Note that the structure H' also forms a tree that is sorted over the identity of the respective trees that represent columns of the table. As in the previous example, each cell is provided with a COUNT parameter, shown as a box next to each cell. As before, the COUNT parameter represents how many elements are present in the branches below the respective element.

Using the structure according to the invention, data can be stored in a database without imposing rigid structures in the form of keys. The structure according to the invention can be expanded, amended and revised by simply adding trees and connections, without compromising performance. The parameters needed for the quick search routine are included in the data structure, and therefore the abovementioned examples of query optimisers can be used without restriction on the database structure. There is no need to build a secondary search database structure.

In the previous example, the elements of a tree are all different. Although this can be the case for some applications, most data to be stored will have multiple identical entries. For this situation, the construction as shown in fig. 5 is used. In this example, the entry Bob occurs three times. During the construction of the data structure, each successive cell for Bob is put next to the cell already there and connected to the others via a pointer ring, or so called self-ring. In the shown example the pointer ring is bi-directional; a unidirectional ring would suffice, but a bi-directional architecture has advantages in the navigation through the ring. The cell that was added latest is directly part of the tree structure. To distinguish between cells of the tree structure that have neighbours in a self-ring and those that do not, each cell is provided with a variable as shown in the lower section of the cells in fig. 5. The value 0 indicates in this case that the cell does not have neighbours, i.e. no multiple occurrences are present. As shown in the fig. 5, each cell within a ring maintains its link to the next tree, as indicated by the dotted lines. In this way the structure fully maintains all data information. Furthermore, the COUNT parameter has to be adjusted for any multiple occurrences due to a ring, so to maintain that the COUNT parameter represents how many cells are present in the branches below the respective cell. The COUNT parameter can also include the number of elements in a self-ring. The self-ring configuration can also be used with the search method shown in the first two examples.

To further make the data structure accessible, all the end elements of the tree structure are provided with linking elements that point towards the upper or start section of the respective tree. Pointers are an efficient way to implement these linking elements. The linking elements provide a ring structure to the trees and make navigating through the structure easier to implement, and in case pointers are used prevent that nil pointers occur.

To implement the data structure use can be made of a data element G according to the invention as shown in fig. 6. This data element can be used universally throughout the data structure, and can be changed to leave out features or include extra features when required. Note that the invention is not limited to this specific data type, and that other implementations can be used.

The data element is shown schematically in fig. 6. The element G is provided with three pairs of pointers and a single pointer. The pointers of the first pair are labelled LVR and RVR (Left Vertical Ring, respectively Right Vertical Ring), the pointers of the second pair are labelled LHR and RHR (Left Horizontal Ring, respectively Right Horizontal Ring), the pointers of the third pair are labelled LSR and RSR (Left Self Ring, respectively Right Self Ring), and the single pointer is labelled IF (Information bridge). The LVR/RVR pair can be used for the tree structure for that incorporates elements of the same type. The LHR/RHR pair can be used to connect an element to neighbouring trees. The LSR/RSR pair can be used to include similar elements into a self-ring structure.

In the initial state as shown in fig. 6 all pointers point to the data element itself. When adding or inserting elements to the structure, the pointers are redirected so that a ring configuration is maintained, so that every pointer in the structure has a valid address, and cases of a non-defined pointer (nil pointer) are avoided. Additionally, the data element can be provided with several parameter values. The IF pointer can be used for connections with any other instance within the data structure; for example an other element of the same or another tree or even elements of one or more levels higher in hierarchy. The IF pointer can be used for example as an InfoBridge; that is a connection element that looks like a Y-adapter. This InfoBridge can be cascaded and/or be bi-directional. With the InfoBridge, any internal data structure can be build within the context of the data structure of the invention.

In this example, only a very small section of a database is shown. In practice, a great number of columns are used (and therefore also a corresponding number of trees), and similar a great number of tables or similar constructions are used. Furthermore, the trees will in the implementation be accessible through some means, for example a linking element such as a pointer, which linking means itself can be part of a further configuration to facilitate access to the tree structures.

The implementation of the data structure and method according to the invention is not limited to the example shown, but can be achieved using any known and suitable manner. Typically, the invention will be implemented as a computer program that is stored in a computer memory or on a data carrier. The program has program code sections that when run on a computer system will perform the steps of the method according to the invention.

The implementation of the system as described above can be made using any known and suitable method and programming language. It is helpful if the language of the implementation supports pointers. It is also useful if the programming language is object orientated, for example C++, which language has the additional benefits of availability of pointers, objects, and object classes. For most implementations additional control structures would be necessary, comprising temporary elements. However, such implementation details as such are known and are within reach of the person skilled in the art.

The data structure according to the invention can be implemented in particularly in a memory that can be randomly accessed (such as for example a memory of the RAM type), wherein the addresses can be randomly accessed. The use of a random access memory also has the advantage that changes to the data structure do not effect efficiency in any way. Although the invention is preferably implemented in a memory with random access, the implementation is not limited to this form, and other implementations in memory devices are possible.

Claims

1. Method for optimising a query with respect to a database structure, comprising:

receiving database information with respect to a specific data source (DS),

generating a search database structure (E) for said data source (DS), based on said received data source information,

receiving a query request for said data source (DS),

analyzing the search database structure (E) by counting a number of hits for respective sections of said query request, and

calculating an optimized query request based on the analysis performed.

2. Method according to claim 1, further comprising

generating said search database structure by organizing data elements of the same type in respective tree structures (A, B).

3. Method according to claim 2, further comprising

providing each data element of a tree with a number (COUNT) representative of the number of data elements arranged in the tree structure (A, B) under said respective data element.

4. Method according to any of the preceding claims, further comprising

linking elements of a first tree (A) with elements of a second tree (B) via linking elements

(H), wherein a linking element represents a data record relation.

5. Method according to claim 4, further comprising determining a key set for the optimized query request.

6. Database structure, for storage of data within a computer system, comprising

data elements of a first type (G), representing database entries, and

data elements of a second type (H), associated with said data elements of the first type (G), wherein the data elements of the first type (G) are arranged in a first tree structure, and wherein the data elements of the second type(H) are arranged in a second tree structure.

7. Data structure according to claim 6, wherein end sections of the tree structures are connected via linking elements to a start section of the tree structure.

8. Data structure according to claim 6 or 7, wherein multiple occurrences of similar data elements of the first type (G) are arranged in a set of data elements that are mutually connected through linking elements, and wherein one data element of the set is directly part of the respective tree structure.

9. Data element for a database structure according to any of the preceding claims 6-8, comprising
a first pointer pair (LSR, RSR),
a second pointer pair (LHR, RHR),
a third pointer pair (LVR, RVR).

10. Data element according to claim 9, further comprising an IF pointer (IF).

11. Computer program product, comprising code portions for executing when loaded into a computer memory the steps of a method according to any of the claims 1-5.

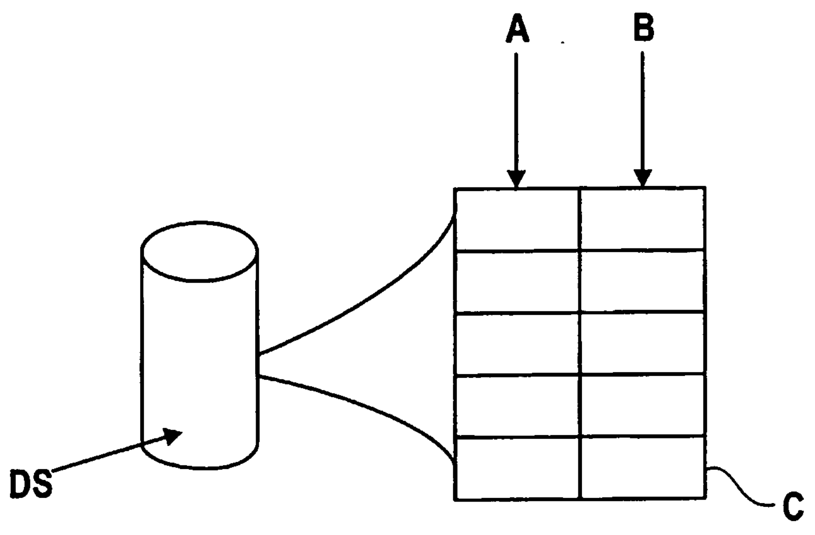


FIG. 1

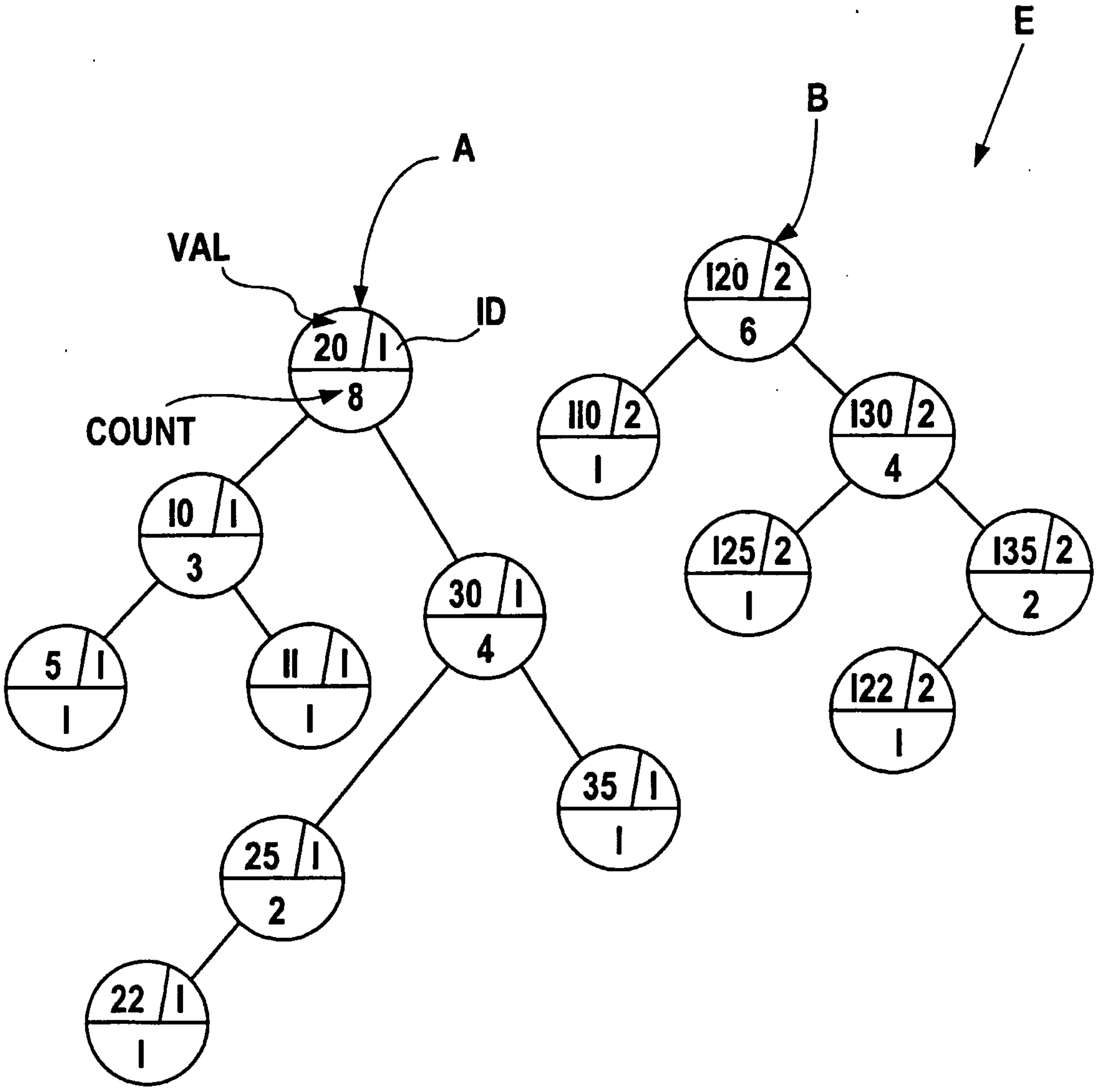

E
B
A
VAL
ID
COUNT
20 / 1
8
10 / 1
3
5 / 1
1
11 / 1
1
30 / 1
4
25 / 1
2
22 / 1
1
35 / 1
1
120 / 2
6
110 / 2
1
130 / 2
4
125 / 2
1
135 / 2
2
122 / 2
1

FIG. 2

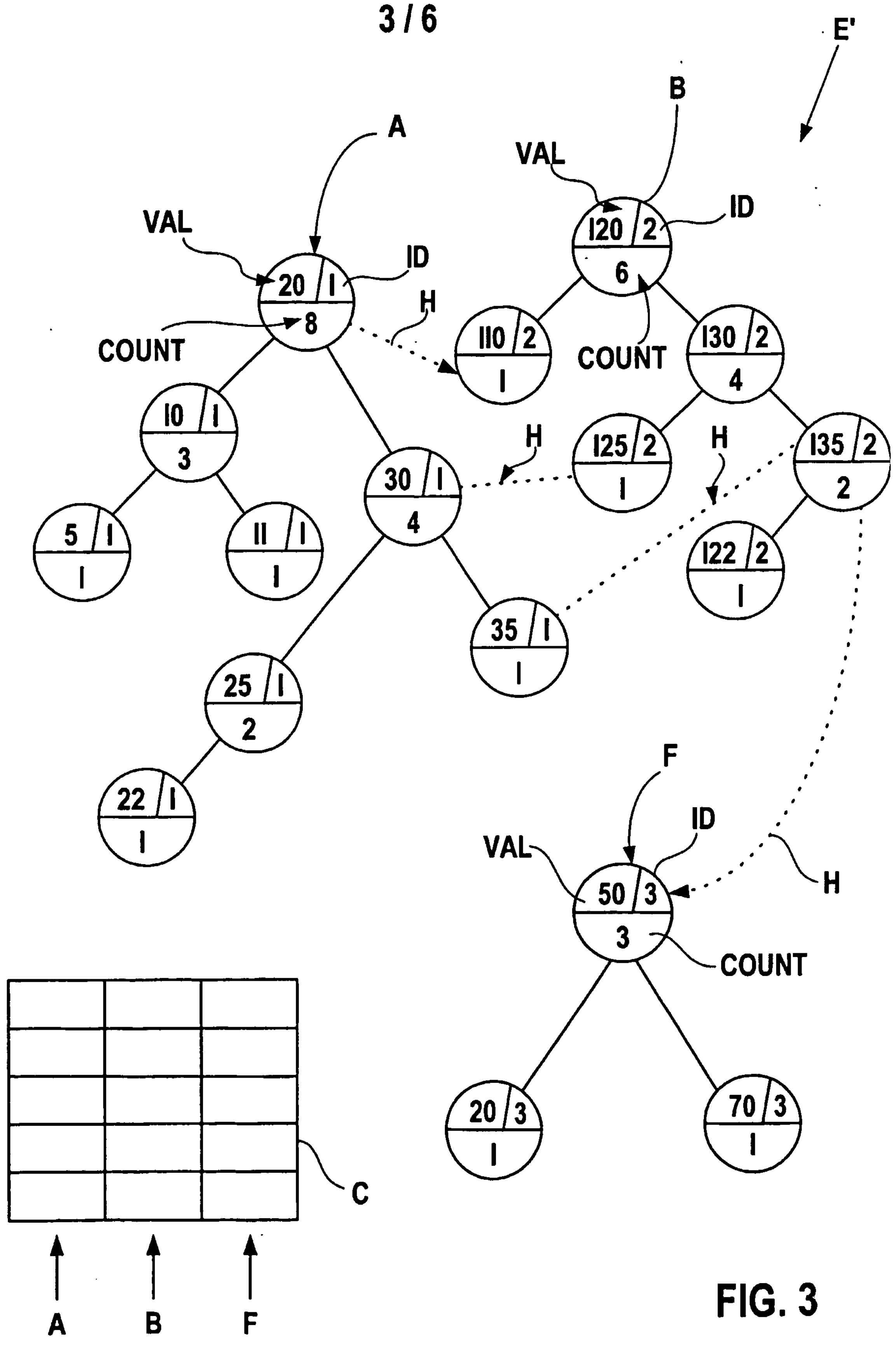
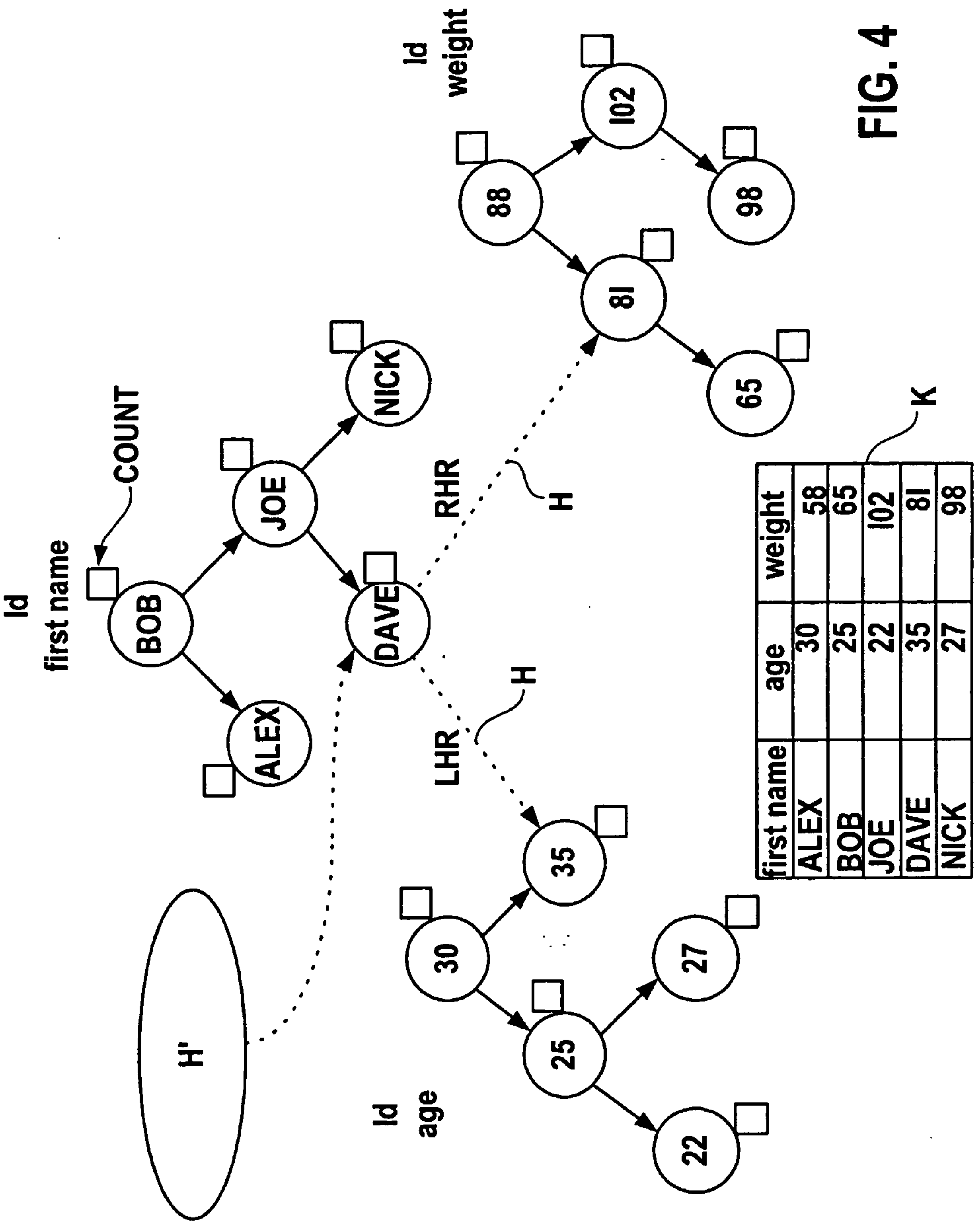

E'
A
B
F
C
VAL
ID
COUNT
H
20 / 1 8
10 / 1 3
5 / 1 1
11 / 1 1
30 / 1 4
25 / 1 2
22 / 1 1
35 / 1 1
120 / 2 6
110 / 2 1
130 / 2 4
125 / 2 1
135 / 2 2
122 / 2 1
50 / 3 3
20 / 3 1
70 / 3 1

FIG. 3



first name	age	weight
ALEX	30	58
BOB	25	65
JOE	22	102
DAVE	35	81
NICK	27	98

FIG. 4

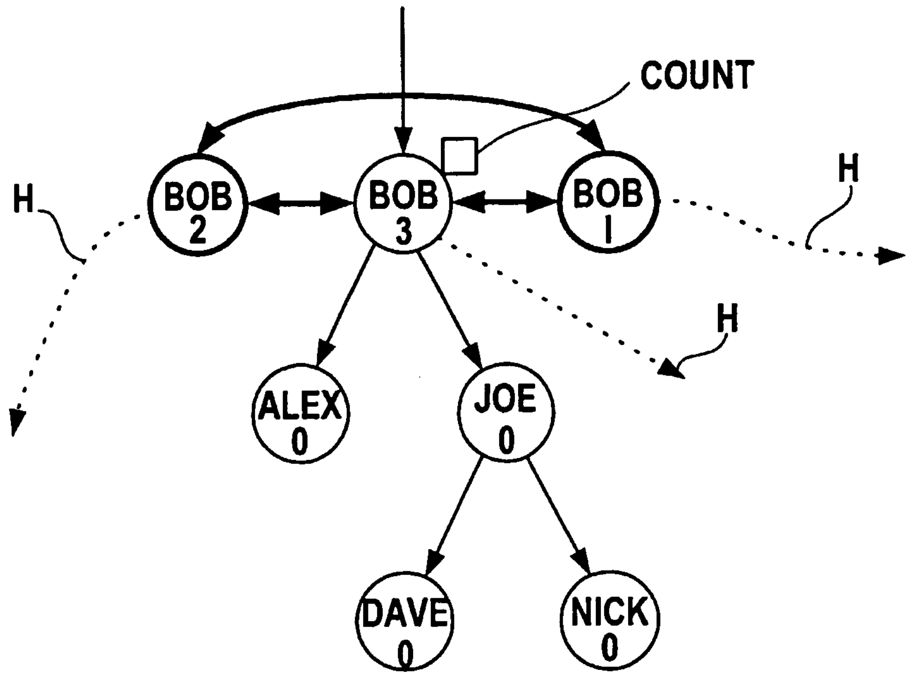


FIG. 5

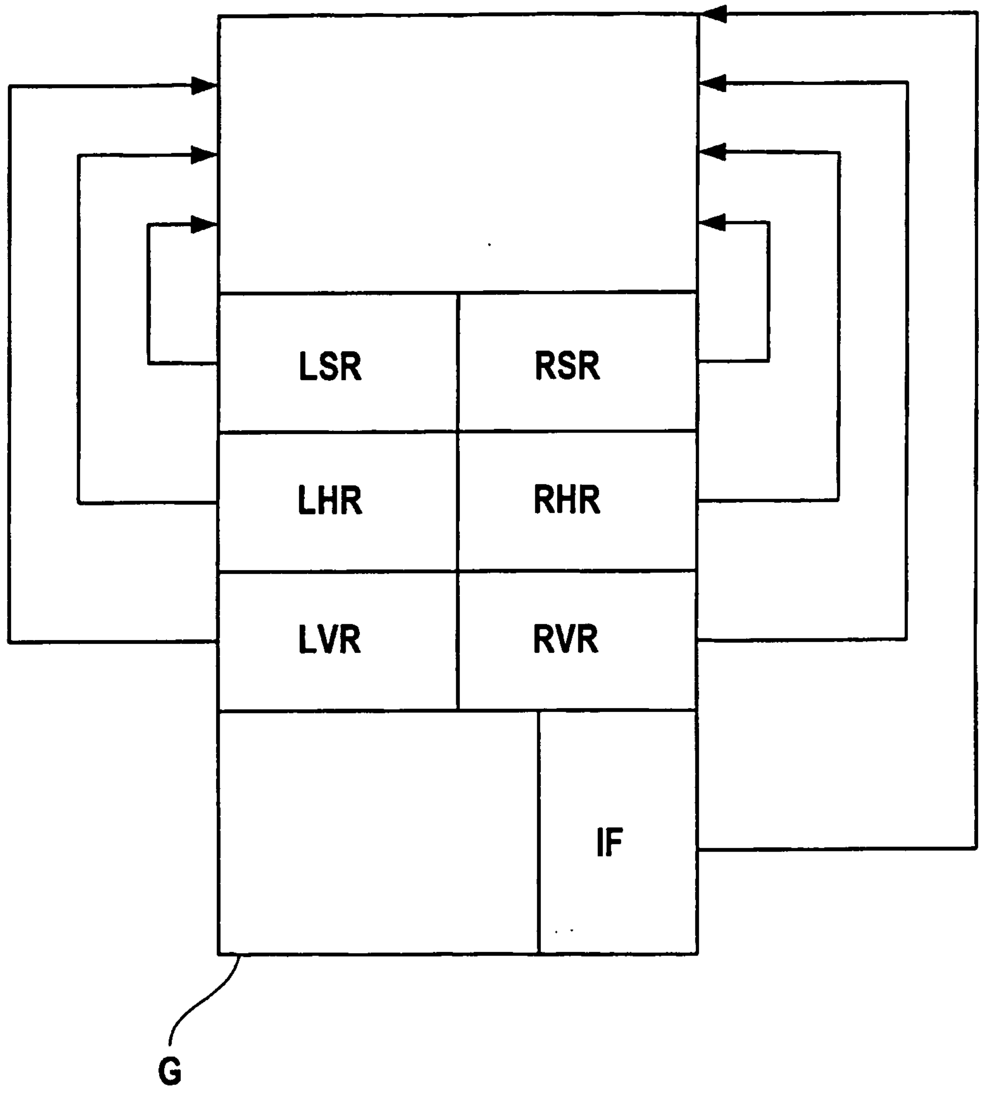


FIG. 6